الجمهورية اليمنية
جامعه صنعاء
كليه الهندسة ـ كهرباء
قسم: تقنيات المعلومات
المستوي: المتوسط



Republic of Yemen
University of Sana'a
Faculty of Engraining
Department (I--T)

بحث بعنوان:

هندسة البرمجيات في اليمن





تحت إشراف الدكتور الفاضل:

د/ علــــــي ألـحـــمـــدي

قسم الكهرباء – تقنيات معلومات

(١١-٣-٢٠٠٧)

إعداد المهندس/

احمد توفيق علي عالم

الفهرس: رقم الصفحة

أولا :

بسم الله الرحمن الرحيم

هندسة البرمجيات

ألمقدمه:

البرمجيات SOFT WARE:

الحاسب الآلي بدون برمجيات كإنسان بلا روح ، كما أن التطور الذي يحدث في أجهزة الحاسب الآلي ومكوناته يصاحبه أيضا تطور وتحديث دائم في عالم البرمجيات .[4].

والبرمجيات بصفة عامة هي عبارة عن مجموعة من الأوامر المرتبة منطقيا ، ويتم تنفيذها بواسطة وحدة المعالجة المركزية للحاسب الآلي ، ويختلف مستوى ونوع البرمجيات طبقا لعلاقاته وقربه من الحاسب الآلي من ناحية ، أو من قربه وعلاقاته بالمستخدم من ناحية أخرى ، فنجد أن نظام التشغيل OPERING SYSTEMS بشكله الأولي هو الملتصق مباشرة بوحدة المعالجة المركزية CPU بينما نجد على الطرف الآخر ، التطبيقات البرمجيةAPPLICATIONS هي الأكثر قربا وسهولة بالنسبة للمستخدم.



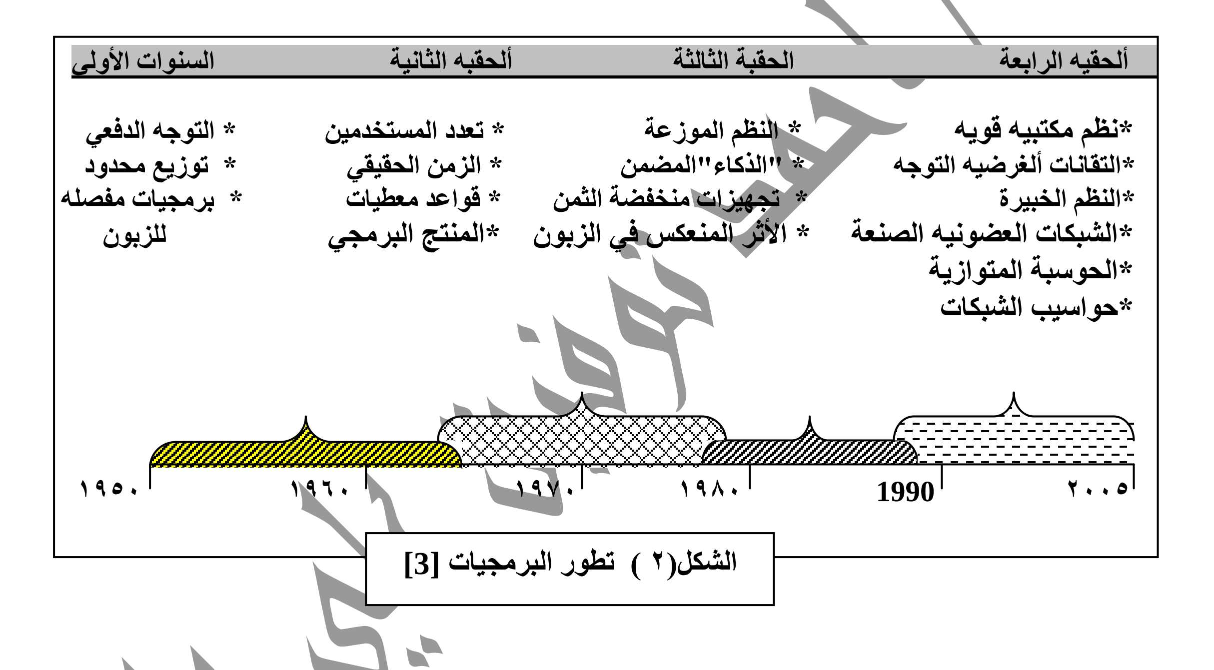
الشكل(١).[1]

هندسة البرمجيات Software engineering هي فرع من فروع المعلوماتية يهدف ألي تطوير مجموعة أسس وقواعد تهدف إلي تحسين طرق تصميم وتطوير البرامج علي جميع المستويات وذلك بطريقه تلبي احتياجات المستخدمين.

وهندسة البرمجيات لا تهتم بكتابة البرنامج نفسه أي بكتابة شفرته بل تحاول تحسين عملية تطوير وصنع البرنامج ابتداء من المواصفات التي يضعها المحترف وانتهاء عند مشكلة صيانة البرنامج أو توسعه.وهي تقوم علي دراسة احتياجات المستخدم وتصميم البرنامج الناسب لها قبل كتابة شفرته، وهناك العديد من الجوانب كالقدرة علي تطوير البرنامج بسهولة لاحقا، أو السرعة، أو إمكانية إضافة ملحقات له بشكل ديناميكي. [2]

١,١ـ تاريخ هندسة البرمجيات:

استخدمت هندسة البرمجيات كمفهوم نظري من حين لآخر في أواخر الخمسينات و بداية الستينات من القرن الماضي. أما الاستخدام الرسمي الأول لهذا المصطلح فكان في مؤتمر عقد من قبل اللجنة العلمية في منظمة حلف شمال الأطلسي ١٩٦٨ حول البرمجيات، وقد أخذ هذا المصطلح بالانتشار منذ ذلك الحين و لاقى اهتماماً متزايداً في نواح مختلفة. عقد المؤتمر لمعالجة مايعرف "أزمة البرمجيات" والتي ظهرت بسبب استخدام الوسائل التقليدية في بناء البرمجيات مما أدى إلى ظهور برمجيات تحتاج إلى وقت كبير لتطويرها، وكلفة مالية عالية أكثر مما هو مخمن لها وكفاءة ضعيفة في إنجاز الوظائف المطلوبة. [3]



الشكل(٢) تطور البرمجيات [3]

١,٢ـ مفهوم هندسة البرمجيات:

البرمجية(**Software**) شيء غير ملموس إلى حد ما بالمقارنة مع المنتجات الأخرى كالعتاد المادي (Hardware). وتمثل البرمجية سلسلة من آلاف أو ملايين الأوامر التي تطلب من الحاسوب إجراء عمليات معينة مثل عرض المعلومات أو إجراء الحسابات أو تخزين البيانات. هذه البرمجيات هي بمثابة الروح من الجسد في النظام الحاسوبي و هي في توسع دائم وازدياد في التعقيد و المتطلبات والمهام التي تقوم بتنفيذها. أما هندسة البرمجيات فهي فرع من فروع الهندسة يقوم على مجموعة أسس و قواعد تهدف إلى تصميم و تطوير البرامج بوفرة ونوعية عالية تلبي احتياجات المستخدمين، هذا الفرع من الهندسة يتميز بأنه لايحتاج إلى رأس مال كبير و بالتالي الخسارة فيه قليلة على عكس بقية الهندسيات، كما لا يكفي لإيجاد البرمجية المتكاملة و الجيدة عمل شخص واحد وإنما يتطلب ذلك فريقاً من المهندسين الجيدين. و قد أصبحت مهنة البرمجة مهنة دارجة جداً يقوم بها كثير من الأفراد غير المحترفين، لذلك كان من الضروري إيجاد هندسة البرمجيات لوضع الأسس والمعايير التي تصون هذه المهنة من المتخصصين بحيث يصبح بالإمكان تمييز البرنامج الجيد من غير الجيد.



الشكل (٣). طبقات هندسة البرمجيات[7]

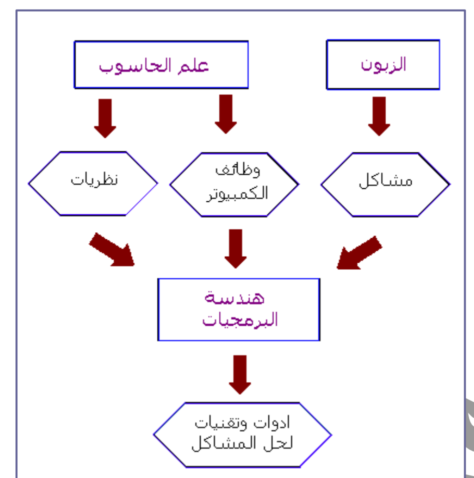
ولفهم **علاقة هندسة البرمجيات بعلوم الكومبيوتر**، يمكن أخذ هذا المثال عن علم الكيمياء واستخدامه في حل المشاكل التي نقابلها في حياتنا اليومية.

يهتم الكيميائي بدراسة المواد الكيميائية (تركيبها وتفاعلاتها، والنظريات التي تحكم سلوكها).

بينما المهندس الكيميائي يستخدم النتائج التي توصل إليها الكيمائي لحل المشاكل التي يطلب منه إيجاد حل لها.

أي أنها من وجهة نظر الكيميائي هو موضوع الدراسة بحد ذاتها.

أما من وجهة نظر المهندس الكيميائي فالكيمياء هي أداة **Tool** تستخدم لا يجاد الحلول لمشاكل عامة (وقد لا تكون هذه المشكلة ذات طبيعة كيميائية بحد ذاتها).



الشكل(٤). علاقة هندسة البرمجيات بعلوم الكومبيوتر.[9]

والفكرة نفسها يمكن النظر ٤ إلى علم الحوسبة **Compute science** حيث يكون تركزينا على الحواسيب ولغات البرمجة لدراستها وتطويرها في حد ذاتها.

أو يمكن النظر إليها والتعامل بها على إنها أدوات نستخدمها عند تصميم وتطوير حل لمشكلة ما تواجهنا أو تواجه الآخرين. [1]

١,٣- الفرق بين البرمجة و هندسة البرمجيات:

تعتبر البرمجة أن كتابة الكود هي أهم عملية في بناء البرامج بغض النظر عن الجدوى من البرنامج أو إمكانية قبول المستخدم له أو حتى قابلية التطوير، في حين أن هندسة البرمجيات تعمل على بناء النظام البرمجي كمشروع متكامل و دراسته من كافة الجوانب: البناء البرمجي، الدعم الفني والصيانة، التسويق والمبيعات، التطوير والتدريب على استخدامه، وبذلك يمكنها بناء الأنظمة الكبيرة لاستخدامها نظام فريق العمل في حين أن البرمجة الفردية تعجز عن ذلك.[1]

١,٤- مهندس البرمجيات software Engineer :

مهندس البرمجيات **software engineer** هو الشخص الذي تقع على عاتقة تطوير المنتجات البرمجية التي تباع للزبائن أو التي يحتاج إليها العملاء والزبائن. ويجب علية إن يتبنى أسلوباً منظماً ونظامياً في عمله لكي يحقق الأهداف المرجوة من استخدام علم هندسة البرمجيات، كما يتحتم علية استخدام الأدوات المناسبة (كاختيار لغة برمجة مناسبة من لغات البرمجة عالية المستوى) والتقنيات الضرورية (كالخوارزميات مختلفة الأعراض)، وذلك كله اعتمادا على نوع المشكلة التي يقوم بحلها وقيود التطوير المفروضة والموارد المتاحة (المادية منها والبشرية).

يقوم مهندسو البرمجيات بجعل الأشياء تعمل، فهم يطبقون النظريات والطرق والأدوات حيثما تكون هذه الأدوات مطلوبة، لكنهم يستخدمون تلك الأدوات بانتقائية ويحاولون اكتشاف الحلول حتى ولو لم تكن هناك نظريات تطبيقية أو طرق دعم. ويدرك المهندسون أيضاً انه يجب عليهم العمل من خلال القيود التنظيمية والمالية، لذلك يبحثون عن الحلول ضمن هذه القيود. ولا يهتم مهندس البرمجيات فقط بالعمليات التقنية لتطوير البرمجيات، ولكنه أيضا يهتم بكل الأنشطة مثل إدارة مشروع البرمجيات وتطوير الأدوات والطرق والنظريات التي تدعم إنتاج البرمجيات.

<u>**أي إن هندسة البرمجيات تعنى**</u> بتصميم وتطوير برامج ذات جودة عالية.

من يشارك في هذه العملية ؟

المشاركون في عملية صناعة البرنامج، عادة ما يندرجون تحت ثلاث مجموعات:

- **الزبون :Customer وهو الشركة (أو الشخص) الممولة لمشروع تطوير البرنامج المطلوب**
- **المستخدم :User الشخص (أو مجموعة الأشخاص) الذي سوف يقوم فعلا باستعمال البرنامج، والتعامل معه مباشرة.**
- **المطور :Developer وهو الشركة (أو الشخص) الذي سوف يقوم بتطوير البرنامج لصالح الزبون . [9].**

الشكل التالي يظهر العلاقة بين الفئات الثلاثة السابقة:

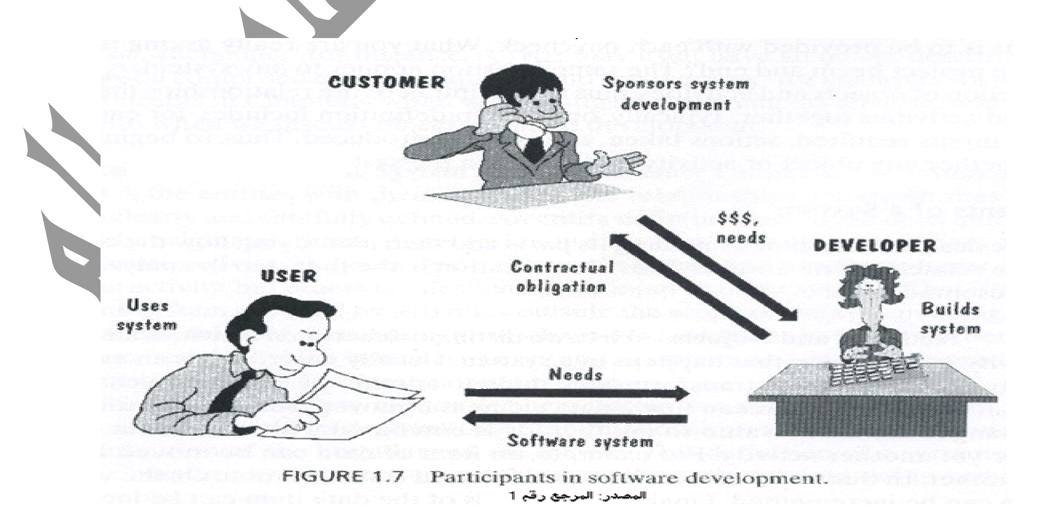


FIGURE 1.7 Participants in software development.
المصدر: المرجع رقم 1

الشكل(٥).مشاركون في عملية البرنامج [9]

*المجالات المختلفة التي له علاقة بهندسة البرمجيات:

- **الرياضيات**: يحتوى أغلب البرنامج على عناصر رياضيات مثل **algorithms** لذالك مطوري البرامج عندهم معرفة بالعديد النواحي الرياضية. الرياضيات تعمل بشكل أفضل أذا كانت صغيرة و هناك صعوبة في حالة الزيادة.
- **العلوم**: البرامج له مواصفات علمية قياسية عديدة مثل الأداء حجم البرنامج تنوع الأحمال.**Network speed** المعادلات الرياضية الطرق الحديثة لقواعد البيانات.
- **علم الهندسة** .
- **التصنيع**: البرامج عبارة عن مجموعة من الخطوات. كل خطوة يتم تحديده وتنفذ بدقة . مثل الكثير من الصناعات، لتحسين وتطوير خطوط الإنتاج والوصول إلى مستوى الجودة المطلوب .
- **إدارة المشروعات**: سواء كان تجاريا أو غير تجارى فأنة يحتاج إلى إدارة. مثل جدول زمني وتكلفة تخصص له . عوامل بشرية للإدارة ومصادر مثل مكتب وأجهزة كمبيوتر.
- **الفن**: واجه (**interface user**) يجب أن تكون ممتعة فنيا للمستخدم. الكود يجب أن يكون ممتع للمبرمج . من القضية الهمة هي أن البرمجة فن.
- **الأداء**: كتابة البرامج تحتاج من المبرمجين إلى استدعاء الطاقة للبحث عن إجابة للأسئلة خلال فترة تواجدهم أمام لوحة المفاتيح.البعض يقول أن هندسة البرمجيات تحتاج أحيانا إلى ألهام. [9]

١,٦ـ مبادئ هندسة البرمجيات :

هناك مجموعة من المبادى والأسس الهندسية **principles of software engineering** والتي من خلال تطبيقات جيداً عند تصميم البرنامج، يمكننا تحقيق الأهداف المنشودة لهندسة البرمجيات.

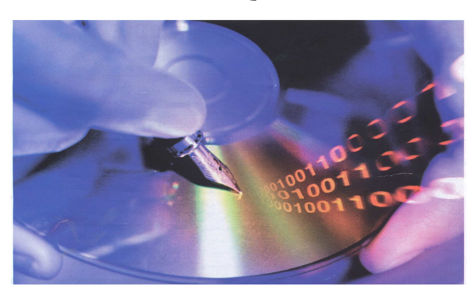
وتتمثل مبادئ هندسة البرمجيات في النقاط الآتية :

١) التجريدabstraction
٢) إخفاء المعلومات information hiding
٣) الوحدوية modularity
٤) المحلية localization
٥) التـجـانـس uniformity
٦) الـتـكـامـلـيـة completeness
٧) الـصـلاحـيـة conformability

١,٧ـ مكونات النظام البرمجيات :

المشاريع التي يتم تطويرها لن تعمل في الفراغ، فعليها أن تتفاعل مع مستخدمين، أجهزة ومعدات متنوعة، نظم تشغيل وبرامج وملفات وقواعد وبيانات..... الخ... وربما حتى أنظمة الحواسيب أخرى.

لهذا يجب تعريف حدود النظام ومكوناته جيداً.
أي يجب تعريف ما الذي يشتمل عليه النظام وما الذي لا يشتمل عليه.



الشكل(٦). مكونات النظام من(١و٠) [1]

أي نظام هو عبارة عن مجموعة من الكائنات **Objects** والنشاطات **activities** بالإضافة على وصف للعلاقات التي تربط تلك الكائنات والنشاطات معا. مع تعريف قائمة المدخلات المطلوبة والخطوات المتبعة والمخرجات الناتجة لكل نشاط . [1]

من مكونات النظام عامه:

- البرامج.
- ملفات التكوين (تسمى أحيانا بالبيانات)، يتم استخدامها لإعداد البرامج.
- مستندات توثيق النظام الذي تصف هيكل النظام، وتحتوي على وثائق تعليمات استخدام النظم، الموجهة لمستخدمي النظام.

أول خطوات تحليل المشكلة هو فهم ماهية المشكلة وتعريفها بوضوح، لذا علينا أولا إن نصف النظام بتحديد مكوناته والعلاقات التي تربط بين هذه المكونات.

١- **النشاطات والكائنات** : النشاط هو عميلة تحدث بالنظام وعادة ما يوصف كحدث يتم من خلال حافز ، النشاط يغير شئ ما إلى أخر بتغيير خواصه(صفاته).
هذا التغير يمكن أن يعنى تحويل احد عناصر البيانات من موقع إلى أخر، أو تعديل قيمته إلى قيمة مختلفة.
هذه العناصر تسمى كائنات objects وهي عادة ما تكون مرتبطة ببعضها البعض بشكل أو بأخر. مثلا الكائنات يمكن إن تكون مرتبة في مصفوفة أو سجل (قيد).
وصف هذه الكائنات نوعها، النشاطات التي يمكن إجراؤها عليها .. يجب وضعها بدقة هي أيضاً.

٢- **العلاقات وحدود النظام Relationships Boundary system** بعد تعريف الكائنات والنشاطات جيدا، يمكن أن نربط بين كل كائن والنشاطات المتعلقة به بدقة، تعريف الكائن يتضمن الموقع الذي سوف ينشأ به (بعض العناصر يمكن أن تكون موجودة بملف سبق إنشاؤه، والبعض قد يتم إنشاؤه خلال حدث ما) والهدف من إنشائه (بعض الكائنات تستخدم من نظم أخرى كمدخلات **Input**)، لذا يمكن التعبير أن لأي نظام حدوداً **boundary** بعض الكائنات بمكن إن تعبر هذه الحدود إلى داخل النظام ، والبعض الأخر هي مخرجات من ذلك النظام ويمكن إن ترحل إلى نظم أخرى .

بهذا يمكن إن نعرف النظام A System على انه تجمع من -:

- مجموعة من الكائنات entities
- مجموعة من الأنشطة activities
- وصف للعلاقات بين الكائنات والأنشطة Relationship .
- تعريف لحدود النظام boundary [2]

١,٧,١ـ أنواع المنتجات البرمجية الأساسية:

هناك نوعان أساسيان من المنتجات البرمجية، هما:

1_ برمجيات عامة شاملةgeneric software :
وهي نظم مستقلة، تنتج بواسطة مؤسسات أو شركات وتباع في السوق لأي عميل، وأحيانا يطلق عليها اسم البرمجيات المغلقة مثل قواعد البيانات ومعالجات النصوص وحزم الرسوم.

2_ البرمجيات الجاهزة (المخصصة)customized software :
وهي نظم مخصصة لعميل معين يطلب تجهيزها، ويتم تطويرها بواسطة شركة أو مطور خصيصا لهذا العميل، ومنها أنظمة التحكم في المعدات الالكترونية والآلات والنظم الخاصة بأعمال معينة.

الفرق الهام بين هذين النوعين من البرمجيات هو إن النوع الأول هو نوع يتم وضع مواصفاته وخصائصه وواجهات الاستخدام ووظائفه بواسطة شركة التطوير، بينما في النوع الثاني يتم فيه التطوير المواصفات حسب طلب الزبون نفسه ومن ثم تقوم شركة التطوير بتنفيذه بناء على طلبات الزبون. [1]

١,٧٤,٢ـ كيف نبني نظاماً ؟

إذا طلب منا عميل تطوير نظام (برنامج) له ، لحل مشكلة معينة تواجهه في عمله ، فمثلا يحتاج نظام حماية لشركة أو نظام صرف إلي لبنك ، أو ممكن إن يكون صاحب مكتبة أو متجر ويريد تغيير نظام البيع والشراء أو العرض ليتم بشكل أفضل .

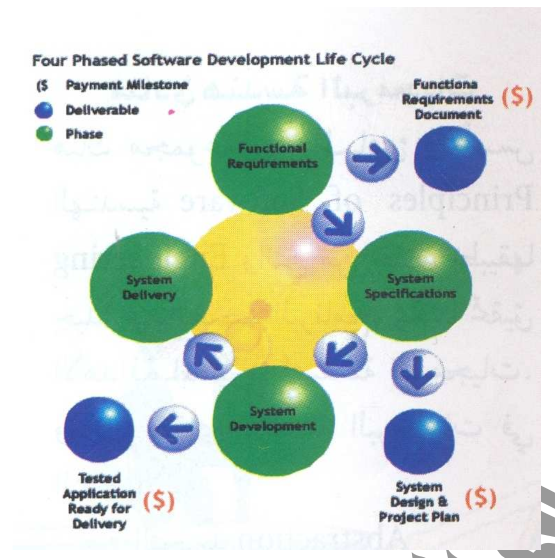
يجب ابتاع الخطوات التالية لبناء هذا النظام:

١. عقد اجتماع من العميل لتحديد متطلباته، هذه المتطلبات تشمل وصف النظام بجميع مكوناته التي شرحنا.
٢. وضع تصميم عام للنظام يحقق المتطلبات التي حددها العميل، وعرضه على العميل ليوضح له الشكل الذي سيظهر عليه النظام عند الانتهاء، ومراجعته معه لأخذ موافقته عليه.
٣. بعد موافقة العميل على التصميم يتم العمل على وضع التصاميم التفصيلية لأجزاء المشروع.
٤. كتابة البرنامج.
٥. اختباره، وإعادة مراجعة المتطلبات التي وضعها العميل للتأكد من تحققها في البرنامج.
٦. تسليم النظام إلى العميل.
٧. بعد تسلم العميل للنظام قد تظهر بعض المشاكل أو الأخطاء التي لم تظهر خلال عمليه الاختبار، والتي تجب على المطور إصلاحها فيما يعرف بصيانة النظام.

[2]

دورة حـيـــاة تـــطـويـر الـنــظـام
Software development life cycle

دورة حياة تطوير النظام هي مجموعة من الأطوار **phases** المترابطة والمتماسكة والنتائج المرتبطة بها) المطلوبة لتطوير وإنتاج نظم برمجية يمكن الاعتماد عليها للقيام بمختلف الوظائف التي آخر، هي العمليات إلي البرمجيات بالشكل الذي يتلائم مع متطلبات الذي يتلاءم مع متطلبات الزبائن وجمهور المستخدمين إجمالا .



الشكل(٧). دورة حياة تطوي النظام [1]

تم تنفيذ غالبية هذه الأطوار بواسطة مهندسي ومطوري البرمجيات. ويجب أن ننوه ونحن بصدد الحديث عن دورة حياة تطوير النظام إلى إن هذه الأطوار قد لا تتطابق بالضبط مع الأطوار المستخدمة في شركة ما. كما إن بعض من العمليات التي تكون مناسبة أكثر من غيرها لبعض أنواع التطبيقات، حيث إذا تم استخدام عمليات لا تتناسب مع طبيعة التطبيق فأنة يتيح عن ذالك منتجات برمجية تتسم بجودة اقل وفائدة اقل . أيضا إن الاسم المحدد المقترن . بكل طور يمكن أن يختلف من شركة إلى أخرى. والأسماء المستخدمة هنا للأطوار المختلفة قد تم اختيارها على ضوء ما هو مستخدم ومتداول بين شريحة واسعة من كبرى الشركات العالمية.[1]

٢,١ـ أطوار دورة حياة تطوير النظام :

١_ تحديد مواصفات النظام specifications phase :

يتم تحليل طلبات الزبون في هذه الطبور وتقدم على هيئة يعرف بـ "مستند المواصفات " الذي يعرض العمليات التي يفرض أن يقوم بها النظام قيد البناء . بمعني أخر، يهتم هذا الطور بمعرفة ما الذي يجب على النظام أن يفعله، وما هي قيود تطويره، كما يعني بتعريف وظيفة **functionality** النظام وقيود العمليات التي يجربها. يسمى هذا الطور أحيانا باسم .

٢_ تطوير النظام development

يتمثل في إنتاج نظام برمحي بحيث يحقق المواصفات المطلوبة .

٣_ التحقيق أو التثبت validation

هي عبارة عن فحص واختيار للنظام للتأكد من إن النظام يحقق ما يريده الزبون بالضبط.

٤_ ارتقاء أو تحديث النظام Evolution:

يتمثل ففي تغير وتحديث النظام كاستجابة طبيعة لمقتضيات التغير .

تقوم بعض عمليات البرمجيات المختلفة بتنظيم هذه الأطوار بطرق مختلفة ووصفها بمستويات مختلفة من التفاصيل، ويختلف توقيت الأطوار كما تختلف نتائج كل نشاط منها.

إن عمليه بناء أي منتج تمر بعدة مراحل يطلق عليها عادة (دورة الحياة) **(life Cycle)_**
فان دورة حياة تطوير أي نظام برمجي تتضمن المراحل التالية:-

١- تحديد وتعريف المتطلبات (Requirements analysis and definition)
٢- تصميم النظام System design
٣- تصميم البرنامج Program design
٤- كتابة البرنامج (تطويره) program implementation
٥- اختبار وحدات البرنامج unit testing
٦- اختبار النظام system testing
٧- تسليم النظام system delivery
٨- الصيانة maintenance

كل مرحلة من تلك المراحل تتضمن العديد من الخطوات أو النشاطات ولكل منها مداخلاتها ومخرجاتها وتأثرها على جودة المنتج النهائي (البرنامج).
دورة حياة أي منتج تبدأ بأول خطوة وهي تحديد المتطلبات وتتدرج إلى باقي الخطوات كما هي مرتبة حتى الوصول إلى أخر خطوة وهي تسليم البرنامج وصيانته (إن دعت الحاجة) ، إلا إن التجارب العلمية تظهر إن هذا ليس ضروريا وان دوره حياة تطوير البرامج قد تأخذ أشكالاً (أو أنماطاً) مختلفة. [2]

٢,٢ـ نماذج دورة حياة تطوير النظام :

النموذج عبارة عن تمثيل مبسط لدورة حياة تطوير النظام حيث تعرض هذه العمليات من منظور خاص.من أمثلة منظور العمليات المستخدمة .منظور مخطط تتابع العمل وتتابع الأطوار، ومنظور تدفق البيانات (تدفق المعلومات)، ومنظور قواعد وأعمال (تحديد أعمال) والنماذج بطبيعتها هي تبسيط لدورة حياة تطوير النظام عبارة عن موجز مجرد للعمليات الفعلية الموصوفة، وقد يحتوي على الأطوار التي هي جزء من عمليات البرمجيات التي ينشغل بها العاملون في هندسة البرمجيات .

١ ـ النموذج الانحداري Waterfall Model:

في هذه النموذج تسير دورة الحياة بشكل تدريجي بدأ من الخطوة (١) وحتى الخطوة (٨). يتميز النموذج الانحداري بالبساطة ، ولذا فانه يسهل على المطور توضيح كيفية سير العمل بالمشروع للعميل (الذي عادة لا يعرف الكثير عن صنع البرمجيات) والمراحل المتبقية من العمل ، وقد كان هذه النموذج أساس عمل كثيرا من المؤسسات لفترة طويل مثل وزارة الدفاع الأمريكية ، واستنبط منه العديد من النماذج الأكثر تعقيداً.
إلا إن لهذا النموذج العديد من العيوب ، أهمها انه لا يعكس الطريقة التي يعمل بها المطورون في الواقع، فباستثناء المشاريع الصغيرة والبسيطة (أي إنها مفهومه بشكل جيد للمطور) فان البرمجيات عادة ما تنتج بعد قدر هائل من التكرار والإعادة ، في حين إن هذا النموذج يفترض أن يكون الحل واضحاً ومفهوماً وسبق تحليله بالكامل قبل مباشرة مرحلة التصميم وهو أمر يكاد يكون شبه مستحيل مع الأنظمة الضخمة ، وحتى إن كان ممكناً فإنه يأخذ وقتاً طويلاً جداً (ربما سنوات)!.
باختصار النموذج الانحداري سهل الفهم وبسيط في إدارته، لكن مميزاته تبدأ في التداعي بمجرد أن يزداد في تعقيد المشروع. [1]

التطوير على مراحل Development Phased:

حسب النموذج الانحداري فانه يجب على المطورين إنهاء مرحلة تحليل المشروع بشكل تام قبل البدء في التصميم ، وكما وضحنا فان هذه المرحلة قد تتطلب وقتا طويلا في بعض المشاريع وقد تمر عدة سنوات قبل إن يرى البرنامج النهائي .
ولكن هل يمكن لسوق العمل الانتظار كل هذه الوقت ؟
الإجابة بالطبع لا .
لذا كان لا بد من إيجاد طرق أخرى لتقليل زمن تطوير المشروع إحدى هذه الطرق هي التطوير على مراحل Development Phased حيث يتم تطوير النظام على عدة مراحل ، بتقديم إصدار من البرنامج به بعض الوظائف للعميل والعمل على تطوير الإصدار الأحق الذي سوف يقدم له بقية الوظائف .

<u>*يوجد عدة طرق يمكن بها تنظيم عمليه تطوير إصدارات البرنامج ، ومن أشهرها :-</u>

<u>٢- النموذج ألتزايدي Incremental model:</u>

حيث يتم تقسيم النظام المطلوب تطويره إلى عدة أجزاء حسب الوظائف التي يتعين عليه القيام بها، يبدأ أول إصدار بأحد تلك الأجزاء ومع الوقت يتم إضافة المزيد من الأجزاء(الوظائف) حتى يتم الانتهاء من تطوير النظام بشكل تام وحسب متطلبات العميل .

<u>٣- النموذج التكراري Iterative model:</u>

هذه المرة يتم تسليم برنامج بكامل الوظائف من أول مرة ولكن يتم تعديل وتغيير بعض تلك الوظائف مع كل إصدار من البرنامج .
من مميزات هذه الأسلوب أنه يمكن المطورين من الحصول على ملاحظات وتقييم الزبون مبكرا وبصورة منتظمة ، ورصد الصعوبات المحتملة قبل التمادي بعيدا في عمليات التطوير كما انه يمّكن من اكتشاف مدى حجم وتعقيد العمل مبكراً.

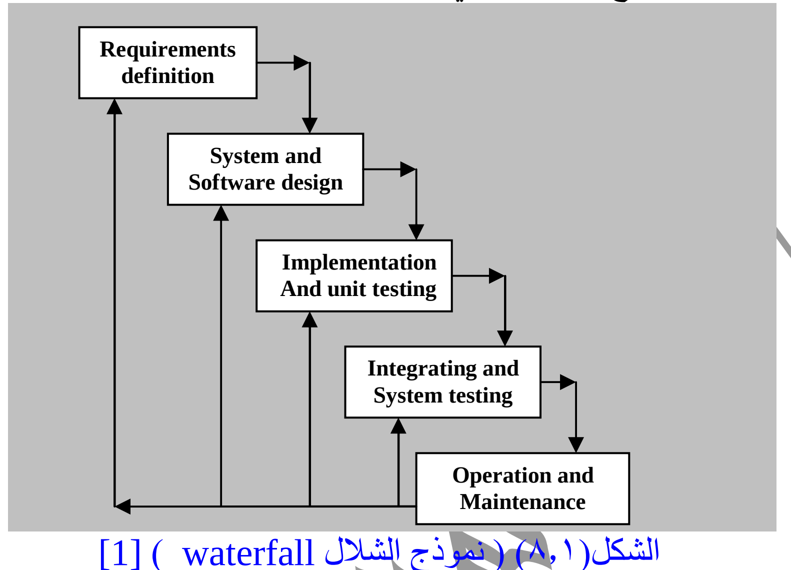
<u>٤- النموذج اللولبي spiral Model:</u>

وهو شبيه لدرجة كبيرة بالنموذج ألتزايدي والتكراري ، ولكن فيه يتم دمج فعاليات التطوير مع إدارة المخاطر **risk** من أجل التحكم بها وتقليلها . يبدأ النموذج اللولبي بمتطلبات العميل مع خطة العمل المبدئية (الميزانية ، قيود النظام ، والبدائل المتاحة) ثم يتقدم خطوة إلى الأمام بتقدير المخاطر وتمثيل البدائل المتاحة قبل تقديم ما يعرف بـ " وثيقة العمليات" **Concept Operations** التي تصف وبشكل عام كيف يجب على النظام أن يعمل ، بعدها يتم تحديد وتدقيق المتطلبات للتأكد من إنها تامة ودقيقة إلى أقصى حد ممكن . بذلك يتكون وثيقة العمليات هي المنتج من الطول الأول، و المتطلبات في المنتج الأساسي من الطور الثاني وفي الطور الثالث تتم عمليه التصميم ، أما الاختبار فيتم خلال الطور الرابع .
في كل طور أو مرحلة يساعد تحليل المخاطر على تقدير البدائل المختلفة في ضوء متطلبات وقيود النظام وتساعد النمذجة على التحقق من ملائمة أي بديل قبل اعتماده.

٥ـ نموذج الشلال (WATER FALL):

نموذج الشلال يمثل أطوار دورة حياة تطوير النظام في مراحل عمليات مستقلة مثل تحديد مواصفات النظام والتصميم والتنفيذ والاختبار والصيانة. ويتم تطوير النظام بإتباع كل مرحلة على حدة ومن ثم الانتقال إلى المرحلة التي تليها. يبين نموذج الشلال تتابع الأطوار في دورة حياة تطوير النظام مع مداخلاتها ومخرجاتها.

ويمكن تمثيل هذا النموذج بالشكل التالي:



الشكل(٨,١) (نموذج الشلال waterfall) [1]

٥,١ ـ إما مراحل نموذج الشلال فتتلخص فيما يلي:

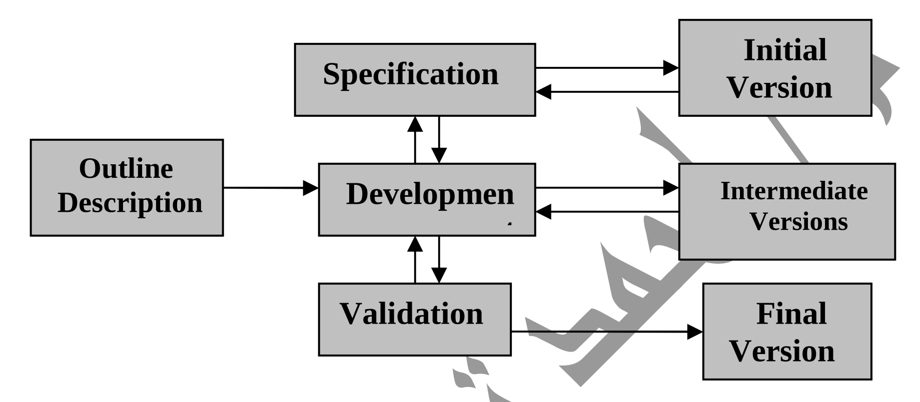
- تصميم النظام وتصميم وحدات البرمجيات
  System and software design
- تنفيذ واختيار وحدات النظام
  Implementation and unit testing
- تجمع النظام واختياره
  Integrating and system testing
- عمل النظام وصيانة
  Operation and maintenance

وينبغي ملاحظة إن هذه المراحل قد تختلف من مطور إلى أخر، كما قد تختلف من شركة إلى أخرى. وقد تتجزأ بعض تلك المراحل إلى عدة مراحل فرعية ، وقد يتم دمج بعض المراحل مع بعضها البعض ،لمنها في إجمالي العمليات لا تخرج عن هذا النطاق .

ومشاكل نموذج الشلال تتخلص في إن التقسيم غير المرن للمشروع إلى مراحل منفصلة يزيد من صعوبة الاستجابة لمتطلبات الزبون المتغيرة ، لهذا يصبح هذا النموذج مفيدا ومرغوباً عند التفهم الكامل لمتطلبات الزبون .

٦- نموذج التطوير الارتقائي (Evolutionary development):

التطوير الارتقائي: هو نموذج تطوير لإنتاج نظم برمجية، تتداخل في هذا النموذج أطوار تحديد مواصفات النظام والتطوير والتثبيت. وفي هذا النموذج يتم وضع خطوط وصف عامة .ثم تتم متابعة العمل في أطوار متزامنة لتحديد المواصفات التي تعطي نماذج أولية التي بدورها لتغيرها أو تعديلها. ثم من طور تحديد . المواصفات تبدأ أعمال التجهيز التي توفر إصدارات وسيطة تتفاعل بدورها مع طور التطوير حيث تؤثر فيه تتأثر أيضا به . ثم يتم التحقيق من النظم إي الطور التحقيق أو التثبت للوصول إلى الإصدار النهائي للنظام. كما تلاحظ أن الطور تحديد مواصفات النظام والتطوير التحقيق أو التثبت تتفاعل مع بعضها البعض.



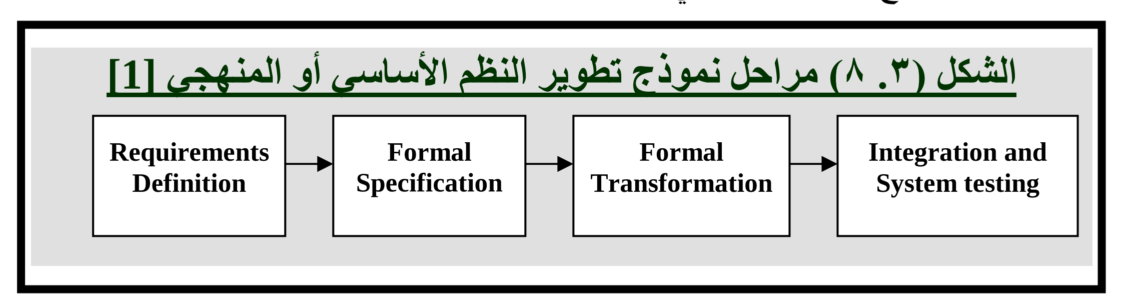
الشكل(٨,٢) (نموذج التطوير الارتقائي) [1]

من المشاكل التي يعاني منها نموذج التطوير الارتقائي عدم وضوح أطوار دورة حياة تطوير النظام، وفقر هيكلية النظام، كما إن هذه النموذج يحتاج إلى مهارات خاصة بلغات برمجية خاصة تسمح بإصدار النماذج الأولية السريعة **Rapid prototyping**.

يمكن تطبيق نموذج التطوير الارتقائي في النظم الصغيرة والمتوسطة الحجم؛ وفي أجزاء من النظم الكبيرة مثل واجهة المستخدم؛ وفي النظم قصيرة الأمد.

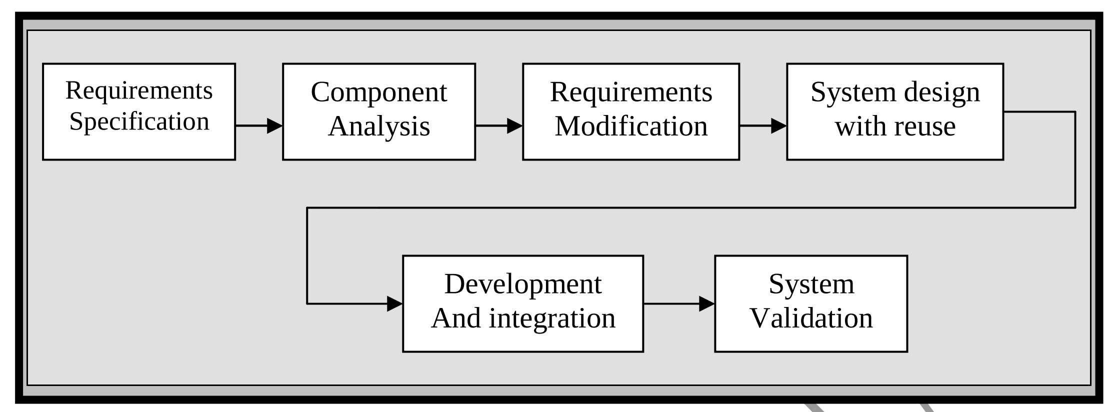
7- نموذج تطوير النظم الأساسي أو المنهجي (Formal system Development) :

يقوم هذا النموذج بتحويل المواصفات الرياضية عبر عدة عمليات إلي برامج تنفيذية، كما إن هذا النموذج يجسد منهجية التطوير النظيف للبرمجيات. تتدرج مراحل هذا النموذج من تعريف المتطلبات (تحديد مواصفات النظام) ووضع مواصفات النظام الأساسية وتحويل أساسي، ثم في الأخير يتم تجمع واختيار النظام تتمثل مشاكل نموذج التطوير الأساسي أو المنهجي في الحاجة إلى مهارات وتدريب خاص، بالإضافة إلى أهناك أجزاء النظام مثل واجهة المستخدم يمكن تطبيق نموذج التطوير الأساسي عالية من الأمان والسرية.


الشكل (٨ .٣) مراحل نموذج تطوير النظم الأساسي أو المنهجي [1]

8- نموذج التطوير المبني على إعادة الاستخدام (Reuse-based development) :

يعتمد هذا النموذج علي إعادة الاستخدام التقليدي؛ حيث يتم تجميع النظام من عدة مكونات موجودة مسبقاً أو من نظم متوفرة.



الشكل(٨,٤) (خطوات نموذج التطوير المبني علي إعادة الاستخدام) [1]

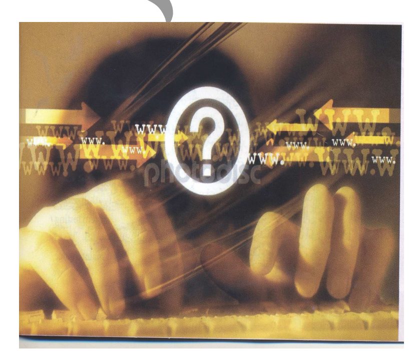
8.1- مراحل العمليات لهذا النموذج هي:

- تحليل المكونات component analysis
- تعديل المتطلبات Requirements modification
- تصميم النظام بإعادة الاستخدام System design with reuse
- التجهيز والتجميع Development and integration

تتمثل خطوات نموذج التطوير المبني على إعادة الاستخدام فيما يلي :

١. توصيف متطلبات النظام.
٢. تحليل المكونات.
٣. تعديل المتطلبات.
٤. تصميم النظام بإعادة الاستخدام .
٥. التنفيذ والتجميع.
٦. التحقق من النظام .

٩- نموذج تكرار العمليات (Process iteration):



الشكل(٨,٥) تكرارا لعمليات [1]

دائماً ما يتم استخراج متطلبات النظام أثناء تنفيذ أطوار النظام لهذا تتكرر عملية الحصول علي متطلبات جديدة - علي مطوري النظام العمل علي تحقيقها لكي يطابق النظام مع متطلبات الزبون التي تم توثيقها في مستند الواصفات – ويعاد العمل علي المراحل المبكرة من الشروع خاصة في النظام الكبيرة .

٩,١- هناك منهجيتان لتكرارهما:

- التطوير المتزايد Incremental development.
- والتطوير الحلزوني Spiral development. [1]

٥- التطوير المتزايد: Incremental development

يعد التطوير المتزايد الخطوة الخامسة من أطوار دورة حياة تطوير النظام، فبدلاً من التواصل لإعداد النظام مرة واحدة يقسم طور تطوير النظام إلي أجزاء متعددة تزيد كلما دعت الحاجة إلي ذلك، ويقوم كل جزء منها بالقيام بوظيفة معينه مطلوبة.
وتكون لمتطلبات المستخدمين اعلي أولوية ، وتوضع المتطلبات ذات الأولوية الأعلى في الأجزاء الأولى من النظام . وما إن يبدأ تطوير جزء حتى يبدأ تجميد المطلبان التي تنتمي إليه وزللك حتى يمكن الاستمرار في الجزء التالي.

٦- التطوير الحلزوني: Spiral development :

عد التطوير الحلزوني المرحلة السادسة من أطوار دورة حياة تطوير النظام، وهو يمثل تطوير العمليات على هيئة حلزون لولبية بدلاً من تتابع متتال الأطوار النظام مع الرجوع عكسياً للتحسين .
وتمثل كل حلقة من الحلزون مرحلة واحدة من مراحل العملية ، ولا توجد مراحل ثابتة تحديد مواصفات النظام أو التطوير ، ويتم اختيار الحلقات في الحلزون بناء على ما هو مطلوب، ويتم تقدير المخاطر risks وتحليلها وحل مصاعبها خلال العمليات .
في الربع الأول من الحلزون يتم تحديد الأهداف **objectives** والبدائل **alternatives** والقيود **constraints**، أما في الربع الثاني فيتم تقييم البدائل وتعريف وتحليل المخاطر.
ويتم التطوير في الربع الثالث والتأكد من منتج المرحلة التالية، وفي الربع الرابع يتم تخطيط المرحلة الثانية.

١٠- نموذج تدفق البيانات (نموذج الأنشطة) :

ويسمى كذلك نموذج المهمة والفعل، ويتم –فيه- تمثيل مهام الأشخاص القائمين بعمليات البرمجيات، وكذلك الأطوار المسئولين عنها.

٣- دراسة متطلبات النظام :

وهو أهم خطوة في تطوير البرامج وهي تحديد متطلبات النظام **Capturing the requirements.**
الهدف من تحديد المتطلبات هو فهم ما يتوقعه العميل والمستخدم من النظام (ما الذي يمكن للنظام أداؤه وما لا يمكنه أداؤه) فقد يكون النظام المطلوب تصميمه بديل لنظام أو لطريقة مستخدمة لأداة مهمة محددة ، أو ممكن أن يكون نظام جديد يقدم خدمة جديدة لم يسبق تقديمها من قبل فلكل نظام برمجي وظيفية معينة ، تحدد بما يمكن له أن يقوم به من جل أداء تلك الوظيفة.
المتطلبات: هي تعريف لشكل النظام أو وصف لما يستطيع هذه النظام أن يقوم به لأداء وظيفته التي سيصمم من اجلها. [2]

٣,١- خطوات تحديد المتطلبات:-

أولاً:- الاجتماع مع العميل للتعرف على المتطلبات :-

وهذه الخطوة هامة جداً إذ أن بقية الخطوات التالية تعمد عليها بشكل أساسي. لذا يجب علينا أن نستخدم كافة التقنيات المتاحة لنكتشف ما الذي يطلبه العميل والمستخدم نبدأ بفهم وتحيل المشكلة التي تواجه المستخدم بكل أبعادها ، نتعرف على العمليات والمصادر التي تتضمنها المشكلة والعلاقات التي تربطها معا ونحدد حدود النظام ، وهذا يمكن إن يتم من خلال:

- طرح الأسئلة على العميل، ومن المفيد أن أحيانا أن نطرح نفس السؤال ولكن بأسلوب مختلف أكثر من مرة فهذا يساعدنا على التأكد من أننا نفهم ما يقصده العميل بالتحديد.
- عرض نظم مشابه للنظام المطلوب سبق تصميمها من قبل.
- تصميم وعرض نماذج لأجزاء من النظام المطلوب أو للنظام بالكامل.

تقسم المتطلبات إلى عدة عناصر تشمل :-

- البيئة المحيطة بالنظام Environment Physical.
- وجهات الاستخدام Interfaces
- المستخدمين وإمكاناتهم Users and human factors.
- وظائف النظام Functionality
- التوثيق Documentation
- البيانات Data
- المصادر Resources
- الأمن Security
- ضمان الجودة Quality Assurance

ويجب التأكد من أن نناقش جميع هذه العناصر.

ثانياً:- تسجيل هذه المتطلبات في وثائق أو قاعدة بيانات ، وعرضها على العميل ليوافق عليها باعتبار إنها ما يطلبه بالفعل:

المتطلبات لا تصف فقط تدفق البيانات والمعلومات من وإلى النظام، وأما تصف كذلك القيود المفروضة على عمل النظام، **وبذلك فان عمليه تحديد المتطلبات تخدم ثلاثة أغراض :**

أولاً:- تمكن المطورين من شرح فهمهم للطريقة التي يود المستخدمين أن يعمل بها النظام.

ثانياً:- توضح للمصممين ماهية الوظائف والخصائص التي سيمتاز بها النظام .

وثالثاً:- توضح المتطلبات لفريق الاختبار ما الذي يجب إثباته لا قناع الزبون أن النظام الذي تم تطويره هو ما سبق أن طلبه بالضبط.

لذلك ولضمان إن كلا من المطورين والزبون متفاهمون تماما على ما يجب القيام به ، فان المتطلبات المسجلة حتى هذه الخطوات **يجب أن تكون لها الصفات التالية :-**

١. أن تكون صحيحة Correct وخالية من الأخطاء
٢. أن تكون ثابتة consistent بمعنى أن لا يكون هناك أي تعارض بين متطلب وأخر.
٣. أن تكون تامة Complete يجب أن يتم ذكر جميع الحالات المختلفة للنظام ، والمدخلات، المخرجات المتوقعة منه .
٤. أن تكون واقعية Realistic بمعنى أن تكون قابلة للتطبيق في الواقع .
٥. أن تكون متعلقة بأمور ضرورة للعميل ، ويتطلبها النظام .
٦. أن يكون من الممكن التحقق منها verifiable.

٧. أن تكون قابلة للتتبع traceable.

يطلق على هذه الوثائق وثائق تعريف المتطلبات **Requirement Definition Document**.

ثالثاً:- إعادة تسجيل المتطلبات بشكل رياضي mathematical ليقوم المصممون بتحويل تلك المتطلبات إلى تصميم جيد للنظام في مرحلة التصميم:

لسنوات عديدة كان يتم الاكتفاء بوثيقة تعريف المتطلبات (التي تحدثنا عنها قبل قليل) والتي تكتب باستعمال اللغة الطبيعية (لغة البشر) لوصف وتسجيل متطلبات النظم بحيث يمكن للعميل أن يفهم كل كلمة موجودة بها، إلا أن ذلك يسبب العديد من المشاكل التي يعود سببها في اغلب الأحيان إلى سوء تفسير بعض التعبيرات للمستخدمين من قبل المصمم أو العكس ، فعلى سبيل المثال قد يطلق المستخدم على النظام التعبير (متوقف عن العمل) إذا كان النظام مشغول بعملية تسجيل احتياطي **backup** باعتبار إن لا يستجيب لأوامر المستخدمة في هذه الحالة ، بينما يعتبر المصمم النظام في هذه الحالة (مستمر في العمل) لأنه يقوم بمهمة أساسية ! لذا فان الاعتماد على اللغة البشرية بشكل تام قد يؤدي إلى أخطاء كثيرة عند تصميم النظام ، وينتج عنها نظام لا يقبله العميل لأنه لا يلبى متطلباته التي حددها من قبل ، ذلك يتم كتابة نوع ثان من الوثائق تسمى (وثائق مواصفات المتطلبات) (**Requirement specification Document**) وهي تكتب باستعمال وسائل وطرق خاصة ابتكرها مهندسو البرمجيات لكتابة المتطلبات بأسلوب تقني بحت . منها على سبيل المثال : لغة النمذجة الموحدة **UML Unified Modeling Language** وهي لغة نمذجة رسمية تقدم لنا صيغة لوصف العناصر الرئيسية للنظم البرمجية .

رابعاً:- التثبت والتحقق من المتطلبات:

التي تم تسجيلها في كلا من وثيقة تعريف المتطلبات (والتي تقدم للعميل) ووثيقة مواصفات المتطلبات (والتي تقدم للمصمم) للتأكد من صحتهما وشموليتهما وان كلا منهما لا تعارض الثانية في إي نقطة ، وإلا فان النتيجة سوف تكون نظام لا يلبى طلبات العميل!

٤ ـ تصميم النظام:

ما هو التصميم ؟

التصميم هو عملية إبداعية لإيجاد حل لمشكلة ، كما تطلق عادة كلمة تصميم على وصف هذا الحل.

حيث نستفيد من المتطلبات التي حددنها في الخطوة السابقة في التعرف على المشكلة ، ثم نبدأ في التفكير في الحل الذي يفي بجميع الشروط والمواصفات التي تحددها المتطلبات ،وغالباً ما يمكن إيجاد عدد غير محدود من الحلول يمكن لنا أن نختار احدها والذي نجده الأنسب من بينها.

عند الانتهاء من خطوة تحديد المتطلبات، فإننا ننتهي بوثيقتين (كما ذكرنا فيما سبق الأولى هي (وثيقة تعريف المتطلبات) ويتم تقديمها للعميل والثانية (وثيقة مواصفات المتطلبات) وتم تقديمهما للمصمم.

ودور المصمم هو تحويل هذه الوثائق إلى نظام يرضي العميل (يلبي احتياجاته) وفي نفس الوقت يرضي المطور (يمكن تطبيقه). [2]

لذا فإن عمليه التصميم في عمليه تكرارية iterative من خطوتين :-

أولا:- يتم إنتاج التصميم التصوري **Conceptual design** والذي يوضح للعميل ما الذي سيقوم به النظام بالتحديد وفي حال موافقة العميل على هذا النظام يتم الانتقال للخطوة التالية .
ثانيا:- تحويل التصميم التصوري إلى وثيقة بها تفاصيل أكثر عن التصميم يطلق عليها اسم التصميم التقني **technical design** والذي يجب أن يظهر للمطور ما هي المعدات والبرمجيات اللازمة لبناء النظام.
أحيانا يتطلب الأمر للعودة إلى الخطوة الأولى (التصميم التصوري) والتعديل عليه ، لذا فأنها عمليه تكرارية حتى الوصول إلى التصميم الذي يرضي العميل ويمكن تطبيقه على ارض الواقع في ظل الإمكانيات المتاحة للمطورين .

١)التصميم التصوري Conceptual design:

يركز هذا التصميم على وظائف النظام **functions** ويكتب بلغة يمكن للعميل أن يفهمها (لغة البشر) ليجيب عن أسئلة العميل حول ماذا(**WHAT**) يعمل النظام ويجب أن يكون خالي تماما من إي تفاصيل برمجية أو فنية ، والأهم إن يحقق كل المتطلبات التي تم تحديدها سابقاً.

٢)التصميم التقني technical design:

هذا التصميم سوف يتم تقديمه إلى مطوري النظام ليقوموا هم بتحويله إلى النظام المطلوب، لذا يجب أن يقدم هذا التصميم إجابة شافية لأسئلة المطور عن كيفية (How) تطوير النظام ، ولمنع إلى تضارب في المفاهيم فإن هذا التصميم عادة ما يكتب باستعمال تعبيرات وأساليب تقنيه .

٥ - كتابة البرنامج واختباره: ***[6:9]***

الجزء الأول: كتابة البرامج :

بعد وضع التصميم للنظام واختيار لغة البرمجة المناسبة، تبدأ الخطوة التي سوف تنقل التصميم المكتوب على الورق إلى واقع. نناقش أهم القواعد التي على المبرمج ابتاعها أثناء كتابة برامجه ، ولكن قبل ذلك نجيب على هذا السؤال التالي:
س: لماذا علينا إتباع هذه القواعد؟
جـ: إذا كان الفرد (المبرمج) يعمل منفرداً في كتابة برامجه، فان أتباعه لقواعد وأساليب قياسية في البرمجة سوف تساعده على تنظيم أفكاره لتجنب الوقوع في الأخطاء كما أنها ستساعده على اكتشاف أي أخطاء قد تحدث بسرعة وبسهولة.

أما آذا كان الفرد يعمل ضمن فريق برمجي ، فان إتباع القواعد والأساليب القياسية في كتابة أجزاء البرامج التي يطلب منه كتابتها، سوف تساعده وبقية الفريق من تنسيق أعمالهم وتنظيمها، كما إنها ستقلل من عدد الأخطاء في البرنامج وتساعد على اكتشاف ما يقع منها في أسرع وقت ممكن .
تفرض الكثير من شركات البرمجة على مبرمجيها إتباع قواعد قياسية في كتابة برامجهم، وذلك ضمان التكامل في جميع البرامج، كما إن بعض الشركات تعين فرق لاختبار البرامج، غير الفريق الذي قام بالبرمجة ولذلك يجب أن يكون الكود البرمجي مكتوب بطريقة واضحة لجميع من يقرأه، وليس لمن قام بكتابته فقط .

٦ـ بعض قواعد البرمجة Programming Guidelines :

لغة البرمجة:

لغة الاله، لغة باسكالBasscal، لغة بيسكBasic، لغة فيجول بيسكVisual Basic، لغة أوراكلOracle، لغة سي بلاس C++/C، C# و غيرها من اللغات .

لغة الآلة MACHINE ANGUAGE : والتي تعتمد على رموز رقمية تكون في صورتها البسيطة من الرقمين(١ ، ٠) لغة تجميعية أعلى قليلا في مستوى تعقيدها وتسمى لغة التجميع ASSEMBLY LANGUAGE ، وتعتمد على رموز حرفية بدلاً من الرموز الرقمية ، كما في لغة الآلة. **أنواع البرمجيات :**

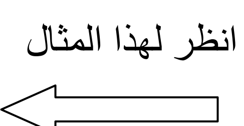
- **نظم التشغيل OPERATTING SYSTEMS**
- **المترجمات COMPILERS**
- **التطبيقات APPLICATIONS-**
- **نظم إدارة قواعد البيانات DATA BASE MANAGEMENT SYSTEM**

٦,١- هياكل التحكم Control Structures :

يقصد بها تلك الهياكل التي تتحكم في مسار عمل البرنامج (مثل Goto ،if- else) وأثناء كتابه هذه الهياكل علينا أن نحاول أن نجعلها واضحة وسهلة التتبع ، وخالية من القفزات الواسعة قدر الإمكان .

```
Benefit = minimum;
if ( age<٧٥ ) goto  A;
Benefit = maximum;
goto  C;
if ( age<٦٥ )goto B;
if ( age<٥٥ )goto;
A: if ( age< ٦٥ ) goto B;
Benefit = benefit* 1.5 + bonus;
goto C;
B: if ( age< ٥٥ ) goto A;
C:  next statement
```

انظر لهذا المثال



نفس الكود يمكن كتابته على هذا النحو :

```
if (age<٥٥) benefit = minimum:
else if( (age<٦٥) benefit = minimum + bonus;
else if (age<٧٥) Benefit = minimum +*15 bonus;
Else Benefit = maximum
```

عالم البرمجة هناك قاعدة تقول أن العمومية ميزة **is a Virtue** generality، لذلك حاول دائما أن تجعل شفراتك البرمجة عامة، لتتمكن من إعادة استعمالها في بقية برامجك بأقل قدر ممكن من التعديل، ولكن حاذر من التمادي في ذلك !

لا تستخدم أبداً أسماء لا معنى لها لمتغيرات أو بارمترات برنامجك

(أريد برنامجاً سريعاً) وكلنا نريد ذلك ولكن ما هو الثمن ؟

عندما تفكر في جعل برنامجك أسرع ما يمكن ، عليك أن تفكر كذلك في الثمن الذي ستدفعه مقابل ذلك :

١. البرنامج السريع قد يتطلب منك كتابة كود معقد يتطلب منك (ومن فريق العمل) المزيد من الوقت والجهد في كتابته.
٢. الوقت الذي تحتاجه عمليه اختبار البرنامج المعقد في مختلف حالته.
٣. الوقت والجهد الذي تحتاجه لتعديل هذا الكود أو لتطويره.

زمن تنفيذ البرنامج ما هو إلا جزء من معادلة كبيرة لحساب تكلفة البرنامج ، لذلك عليك أن تعادل بين سرعة، والجودة واحتياجات الزبون ،ولاتضحى بالبساطة والوضوح من اجل السرعة. [1]

الجزء الثاني: اختبار البرنامج:

وصلنا ألان إلى أخر مرحلة من تطوير النظام ، وهي اختبار البرنامج للتأكد من أنه يعمل على النحو الذي يتوقعه الزبون .

قبل تسليم النظام النهائي إلى الزبون تجرى عليه الكثير من الاختبارات ، بعضها يعتمد على ما الذي يتم اختباره مثلاً:

(احمد مكونات البرنامج – مجموعة من المكونات – جزء من النظام – النظام بالكامل).

والبعض الأخر يعتمد على ما الذي نريد معرفته من هذه الاختبارات مثلا:-

- هل يعمل النظام وفقا لما ورد في المتطلبات ؟
- هل يعمل النظام وفقا لما ورد في التصميم ؟
- هل يعمل النظام كما يتوقعه الزبون منه ؟

مراحل الاختبار:-

عند العمل على اختبار نظام من الحجم الكبير ، فان عملية الاختبار تتم على عدة مراحل موجزها في ما يلي:-

١. اختبار المكون component testingأو Module testing:

أول مراحل اختبار النظم ، هي اختبار كل مكون على حدة بمعزل عن بقية مكونات النظام ، للتأكد من عمله على النحو المتوقع منه ، باختبار المعلومات المتحصل عليها (**Output**) منه بعد إمداده بالبيانات اللازمة له (**input**).

٢. اختبار التكامل integration:Testing:

بعد اختبار كل مكونات النظام والتأكد من سلامة تصميمها ، يجب أن نتأكد من أنها ستعمل معا بشكل صحيح وانه لا يوجد تضارب بين بعضها البعض بحيث أن المعلومات المنقلة بين هذه المكونات تصل بالهيئة المتوقعة لها ، وهذا هو الهدف من اختبار التكامل .

٣. اختبار الوظيفية Function:

ويقصد به اختبار النظام بعد تجميع كل مكوناته للتأكد من أنه يؤدى الوظيفية التي يتعين عليه القيام بها ، والموضحة في وثائق متطلبات النظام ، عندما يجتاز النظام هذا الاختبار يمكننا اعتبار هذا النظام على انه نظام عامل **Functioning system**.

٤. اختبار الأداء performance testing:

في هذه الخطوة يتم اختيار أداء البرنامج في بيئة عمل الزبون للتأكد من أن النظام متوافق مع بقية المتطلبات عند اجتياز النظام لهذا الاختبار يتم التصديق على النظام **Validated system** وبهذا فأننا نعتبر إن النظام أصبح جاهز حسب مفهومنا لما طلبه الزبون.

٥. اختبار القبول Acceptance test;

يتم إجراء هذه الاختبار للتأكد من أن النظام المحقق موافق لما توقعه الزبون ، وبعدها يعد النظام مقبول عند المستخدم والزبون **Acceptance test**

٦. اختبار التثبيت Installation test:

الاختبار الأخير يتم فيه تثبيت النظام في بيئة العمل الخاصة به والتأكد من انه يعمل كما هو مطلوب منه.

٧-لغة النمذجة الموحدة :

طريق البرمجيات هي منهجية لتطوير البرمجيات بغرض تسهيل إنتاج برمجيات عالية الجودة بطريقة اقتصادية.

كانت الطرق الأولى ظهرت في السبعينات من القرن الماضي، تمثل في التحليل الهيكلي **DeMarco 1978 structured analysis**)) وطريقة((**JSD Jackson 1983**)وحاولت هذه الطرق تعريف المكونات الوظيفة لبرامج ما وما زالت هذه الطرق الموجهة بالوظائف (الوظائفية المنى) **function- oriented methods**مستخدمة على نطاق واسع. وفي الثمانينات وتسعينات القرن الماضي ثم استكمال هذه الطرق بالطريق الكائنة المنحى **object- oriented methods** وتجمعت هذه المنهجيات المختلفة في منهجية واحدة موحد مبنية على لغة النمذجة الموحدة **unified modeling language uml** وقد بنيت كل هذه الطرق على فكرة نماذج تطوير

الأنظمة تمثلها بالرسم ، وباستخدام هذه النماذج كمواصفات نظام **specification**، أو تصميم **design** ، يجب إن تحتوي الطرق على عدد من المكونات المختلفة .

ويجب التنويه إلى أنه لا توجد طريقة نموذجية معينة، إذ إن لكل طريقة من الطرق سالفة الذكر مجالات مختلفة لتطبيقها. وكمثال على ذلك فان الطرق كائنيه المنحى تكون مرغوبة في النظم التفاعلية **interactive systems**، لكنها ليست كذلك في نظم متطلبات الوقت الحقيقي الحرجة **stringent real- time requirements.**

[1]

٨- *هندسة البرمجيات بمساعدة الكمبيوتر CASE:*

يشير مصطلح case إلى العبارة التالية **computer-aided software engineering)** الذي تعني هندسة البرمجيات بمساعدة الكمبيوتر. هندسة البرمجيات بمساعدة الكمبيوتر عبارة عن نظم برمجية توفير دعماً آليا لأطوار دورة حياة تطوير النظام، كما تستخدم نظم **CASE** لدعم طرق ومنهجيات هندسية البرمجيات.

تقنية **CASE** تغطي نطاقاً واسعاً من أنواع البرامج المختلفة التي تستخدم في دعم أطوار دورة حياة النظام مثل تحليل المتطلبات **requirements analysis system modeling** ونمذجة النظام **system modeling** واكتشاف وتصحيح العلل والأخطاء **debugging**، والاختيار **testing** .

وترتبط كل طرق هندسة البرمجيات حالياً مع تقنية **CASE**، مثل المحررات التي تستخدم لتسجيل الملاحظات (التدوين) **notations**، ووحدات التحليل **analysis modules** التي تختبر نموذج النظام بناء على قواعد الطريقة المستخدمة، ومولدات التقارير التي تساهم في إنشاء توثيق النظام **system documentation.**

وقد تحتوي **CASE** أيضا على مولد شفرة **code generator** الذي يقوم أليا بتوليد شفرة المصدر **source code** من نموذج النظام. كما يمكن إن يحتوي **CASE** على بعض موجهات البرمجيات عما سيفعله في الخطوة التالية. [1]

٩ - *خصائص البرمجيات الجيدة :*

تسعى طرق ومنهجيات هندسة البرمجيات إلى إنتاج نظم برمجية تحقق مجموعة من الخصائص المرتبطة بالبرمجيات والتي تعكس جودتها. ولا تتعلق هذه الخصائص بما تفعله البرمجيات مباشرة، بقدر ما تعكس سلوك هذه البرمجيات عند تنفيذها، وأيضا بدر ما تعكس من هيكل تنظيم البرنامج المصدر والتوثيق المرتبط بهذه البرمجيات. وأحيانا تسمى هذه الخصائص بـ السمات غير الوظيفية **(non – functional attributes).** ومن هذه الخصائص الأتي:

i. قابلة الصيانة maintainability:

تعني أنة يجب إن تقبل النظم البرمجية احتياجات التغير. أي أنة يتم كتابة البرنامج بطريقة تلبي احتياجات تغيير يحتاجها الزبون أو المستخدم. وهذه خاصة حرجة لتغيرات البرامج نتاج حتمي لتغيرات بيئة العمل.

ii. الموثوقية dependability:

تعني ضرورة إن تكون البرمجيات جديرة بالثقة (أي موثوق بها) **trustworthy.** وهذه الخاصية تشمل على نطاق واسع من الخصائص، التامين **security**، والأمان **safety**، الاعتماد عليها **reliability**، فالبرمجيات التي يعتمد عليها لا تسبب ضرراً تلفاً فيزيائياً أو اقتصادياً في حالة حدوث انهيار للنظام.

iii. الكفاءة efficiency :

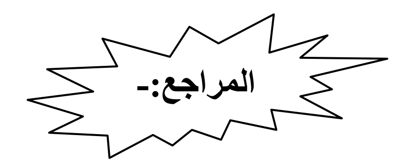
تعني إلا يكون هناك إهدار لموارد النظام **system resources.** حيث أنة لا يجب على البرامج إهدار موارد النظام مثل الذاكرة ودورات المعالج **processor cycles.** هذه يعني كفاءة النظم تتضمن : الاستجابة **responsiveness** ، ووقت المعالجة **processing time** ، واستخدام الذاكرة الأمثل **memory utilization** وغيرها .

v. قابلة الاستخدام usability :

تعني استخدام البرمجيات بواسطة المستخدمين لما هي مصممة له. بمعنى آخر، إن تكون البرمجيات قابلة للاستعمال بدون مجهود لا مبرر له، وهذا يستلزم وجود واجهة مستخدم مناسبة تحتوي على مستندات التوثيق الكافية للرجوع إليها عند الطلب.

iv. قابلية فهم شفرة البرنامج understandability of program code:

وفي الأخير ننوه إلى إن مجموعة الخصائص التي يجب توقعها من أي نظام برمجيات تعتمد بديهيا على التطبيق ذاته ، لهذا فان النظم البنكية على سبيل المثال يجب إن تكون مؤمنة **secure** ، بينما نظم الاتصالات الهاتفية يجب إن تكون موثوقاً بها يعتمد عليها **reliable** ، إما برامج الألعاب فيجب إن تكون حساسة التفاعلية **responsive interactive.** إن مهندسي البرمجيات الذين يقومون بعمل التحليلات وتحديد الخصائص وتصميم وتطوير واختيار وصيانة البرمجيات ، هم في الحقيقة المسئولون عن تحقيق وتطبيق الخصائص التي تلائم المنتجات البرمجية التي يبنونها. [1:2]



١- بشير علي علي النصيف(٢٠٠٦): "هندسة البرمجيات"، مجلة تكنولوجيا الاتصالات والمعلومات – العدد(٦١) يوليو٢٠٠٦م- تصدر عن وزارة الاتصالات وتقنية المعلومات-اليمن، ص[١٤-٢٢].

٢- ماجد محمد الصبا ري(٢٠٠٦): " هندسة البرمجيات"- مجله الحرس الجمهوري –العدد (٢٠) أكتوبر ٢٠٠٦م- تصدر عن قيادة قوات الحرس الجمهوري - ص [69-77] .

٣- د/عبد الواسع العزاني(٢٠٠٦): "هندسة البرمجيات"- مركز الحذيفي –أمام كلية العلوم- جامعة صنعاء- الجمهورية اليمنية.

٤- د/علي ألحمدي (ديسمبر- ٢٠٠٦):"محاضره تقارير"-كلية الهندسة- جامعة صنعاء- الجمهورية اليمنية.

5- Shari Pfleeger,: software – Engineering – Theory and practice, 2nd Edition

*مواقع انترنت، عن موضوع هندسة البرمجيات:

6- http://www.yemensoft.com 2/12/2006
7- http://www.rspa.com ----/1/2007
8- http://www.google.com/ software engineering 9/11/2006

